NUCLEAR UTILITY YEAR 2000 READINESS

Appendix F

TEST SPECIFICATIONS

EMBEDDED SYSTEMS TESTING

1. INTRODUCTION

In any Year 2000 project, testing is perhaps the key element of the project. There are two distinct phases in testing.

The first phase is ***Investigative testing*** to ascertain whether a software program, product or integrated system complies with a predetermined set of specifications for the Year 2000.

The second phase is ***Post remediation testing*** to establish that any modifications made as a result of errors found either in the first phase of testing or by other analytical methods, are valid and the system, product or program can be certified to comply with the Year 2000 specification.

Year 2000 compliance may be defined differently for different purposes. In addition, local installations will vary in the way dates and times are formatted and represented. These differences notwithstanding, the kind of compliance testing that needs to be performed can be categorized into date and date-and-time functionality testing.

Most "traditional" business processing environments are concerned more with the date-category than with the date-and-time-category of functionality. In an Instrumentation & Control environment, the date-and-time-category functionality takes on more importance because of the "hard" real-time requirements of process and device control, monitoring, event signaling etc. performed by embedded systems. Of course, some functionality in these systems is centered on (real) time, with no date-related requirements. Since this functionality is not considered a year 2000 compliance issue, it is not addressed here.

This document presents a comprehensive set of test guidelines and methodology for the testing of embedded systems. In the first section, some generally accepted standards and a brief background of date and time notation is laid out. This is followed by a consideration of the unique challenges presented by the embedded systems. Then a set of specifications based on industry standard guidelines is set out. This forms the basis from which test parameters and methods are drawn.

Testing strategy, which includes precautions, preparations, and considerations of functionality, is then explained. This is followed by the detailed test procedures.

Finally extracts from industry sources such as the IEEE and the NIST along with references for further reading are included in the appendix.

2. BACKGROUND OF DATE AND TIME NOTATION

2.1 *International calendar*

The international calendar currently followed is almost all countries is the ***Julian calendar*** with the Gregorian correction, or simply called the ***Gregorian calendar.***

This is a solar calendar i.e. a year is based on the time taken for the earth to revolve round the sun. It consists of 12 months in a year. Each month consists of a specified number of days. Only the second month February consists of 28 days in common years and 29 in leap years. Thus common years have 365 days and leap years have 366 days.

2.2 ***Definition of a leap year***

A leap year is a year where is an extra day (i.e. February 29th). This intercalation of day is to adjust for the discrepancy arising out of the normal year period of 365 days and the actual solar year based on the earth's revolution which is 365.242199 (365 days, 5 hours, 48 minutes and 46 seconds).

2.3 ***How is a leap year determined?***

As per the Julian Calendar every year divisible by four was a leap year. This led to a discrepancy of 3.12 extra days over four centuries. Pope Gregory corrected this in 1582. As per the correction century years are leap years only if they are divisible by 400. There was also a further refinement, which stated that years divisible by 4000 are common years or non-leap years. With these refinements, the discrepancy between the calendar time and the actual solar time is reduced to one day over four thousand years.

As per the current Gregorian calendar the determination of a leap year is as follows:

1. All non-century years divisible by four are leap years.
2. All century years divisible by 400 are leap years.
   This means that 1900 and 2100 are not leap years, while 2000 is a leap year.
3. As per the refinement to the Gregorian calendar we also have the additional clause: All years divisible by 4000
   are common years or non-leap years.

2.4 ***Julian representation of a date***

The Julian representation of a date is the format DDD, YY or DDD, YYYY where DDD is a number from 1 to 365 or 366 depending on whether the year was a leap year or a common year and YY or YY is the two digit or four digit representation of the year.

2.5 ***Gregorian representation of a date***

The Gregorian representation of a date is the format DD/MM/YY, MM/DD/YY or any of the other common formats currently used that incorporate date, month and year.

In a system ,dates may be stored in Gregorian or Julian representation, or a combination of both. There are also situations where all internal representation and calculations are done using the Julian representations and all external interfaces and displays use the Gregorian representation.

3 THE CHALLENGE OF EMBEDDED SYSTEMS

Embedded systems pose many challenges for testing and remediation of the Year 2000 problem. These can be broadly categorized as follows:

3.1 Architectural

- There is a wide prevalence of four bit and eight bit processors such as those manufactured by Intel, Zilog and Advanced Micro Devices. Many of these have a limited instruction set. Many of these microcontrollers have a two-digit date representation for arithmetic and logical operations.

- Date representation may be different for 'power on ' conditions and in battery backup condition

- There is no standard way to encode dates between different vendors.

3.2 Programming

- Source code is not available for many of these systems.
- Object code may be stored in different levels of firmware i.e. Programmable logic arrays (PLA's), Flash ROM, CMOS or BIOS.
- Object code may be hard coded, reloadable or re-entrant.
- Program may be recording real time intervals based on calendar dates rather than actual dates themselves.

3.3 Configuration

- System may be consisting of upstream and downstream devices that have data interfaces between them.
- Downstream devices may have dates that are set or overridden by upstream devices.
- System may have external interfaces that transmit and receive date information.

3.4 Operational

- The system may be in a production environment that it cannot be taken out of without severe impact.
- Backup systems may not be available, in case of failure during testing.
- Many systems may not revert back to current dates after dates are advanced during testing.
- Warranties, inspection and service logs may be voided by date advancement.

4 SPECIFICATIONS FOR CENTURY COMPLIANCE

The rules that follow are taken from the following source:

http://www.year2000.com/archive/gte-article/NFgte-table3.html

They resemble but are not identical with the rules issued by BSI/DISC. In particular the BSI/DISC rules explicitly cover the point that Year 2000 is a Leap Year. However the rules below have been cited by a English lawyer as a possible standard; and given the source, they might be assumed to be a de facto standard for North America. These rules are also currently being studied by the IEEE as the framework for an IEEE specification on Century Compliance.

4.1. General integrity

No value for current date will cause interruptions in normal operation. As a system date advances normally on a system, each system date must not lead to erroneous operation of the system or its software processes. The best

recognized high-risk date change is the roll over to 2000. However there are a number of high risk dates such as 9/9/99, 2/28/00, etc. which must also be considered.

4.2. Date integrity

All manipulations of calendar-related data (dates, durations, days of week, etc.) will produce desired results for all valid date values within the application domain.

4.3. Explicit century

Date elements in interfaces and data storage permit specifying (i.e. specification of the) century, to eliminate date ambiguity. This criterion essentially requires the capability to store explicit values for the century. It must be noted that this must be interpreted as applicable to embedded systems. Not all embedded systems and their component microcontrollers will have this capability.

4.4. Implicit century

For any date element without century, the correct century is unambiguous for all manipulations involving that element. This last criterion requires that if the century is not explicitly provided, its value can be correctly inferred with 100% accuracy from the date provided.

Although the four criteria defined above fully define century compliance, it must be noted that compliance represents a balance between cost and risk rather than an absolute measure. The application of these criteria will vary depending on the system, the criticality to the line of business, the availability of the system for testing and certification, and the test process itself.

5 TESTING STRATEGY

The testing strategy can be divided into several areas:

5.1 Test Parameters.

Based on the compliance criteria defined above, each individual device or system must be studied to determine the characteristics of the device that will certify functionality. It must be noted that not all the functional characteristics of the device need be tested, such as real time functionality or other characteristics that do not have a time related impact.

To illustrate the kind of functionality, from which testbeds can be drawn, testing can be further categorized by functionality as shown below. These examples are not exhaustive of the kind of functionality found in each category. Subject matter experts should be used to determine what date and time related functions need to be tested for a given device or system.

<u>Conversion and Extraction Functionality</u>

The kind of routines to be tested here include such functionality as:

DayOfYear (YYYYMMDD). This kind of routine might be invoked in systems where dates are represented at some point inside a program using the Julian date format. For example, dayOfYear (20000101) should return 1, whereas dayOfYear (20000229) should return 60. Error conditions are candidates for testing here too. For example, dayOfYear (20010229) should *not* return 59, 60, 61 or any other number, as the input is *not* a valid date. Testing for this kind of condition may be difficult, since a fully year 2000-compliant system should *not* allow the system date to be set to an invalid date!

Conversely, routine such as **date (YYYYJJJ)** and **month (YYYYJJJ)** should correctly convert to Gregorian equivalents of Julian dates. For example, date (1999365) should return 31 and month (1999365) should return 12, or 'DEC' or 'DECEMBER,' depending on the system requirements. Similarly, date (2000060) should correspond to February 29th, *not* March 1st. As before, although testing may be difficult, error conditions should be detected and handled; for example, 19990, 1997-10, 2004366, 2020367 etc.

Systems may differ in terms of whether they represent date and time information independently of each other, or compounded into some kind of timestamp structure. (Some systems may use both representations for different purposes.) In the case of compounded, timestamp representations, routines similar to these may be defined over inputs of the form **YYYYJJJhhmmss** or **YYYYMMDDhhmmss**. In real-time systems, representations may typically be defined to greater levels of precision than seconds.

Depending on how a system boundary has been drawn, date and/or date-and-time *formatting* may need to be tested. Date and/or time outputs may appear on terminals, printers, LED and LCD displays, analog meters, digitally simulated analog meters etc. Even if date and/or time data does not display directly, it may be used to derive or calibrate data that is displayed on these types of device, or data that is used for annunciation. These systems should be validated for year 2000 compliance through to the data display portion of the system boundary, particularly if some critical operator intervention might depend on the accuracy of the data.

Arithmetic Functionality

The kind of routines to be tested here include such functionality as:

daysBetween (startDate, endDate). This kind of routine might be invoked on a regular basis by software that calculates inspection, maintenance, replacement schedules etc. or that statistically analyzes raw data. Year 2000 compliance testbeds should test for correctness of cases such as days between (19991231, 20000301), which should calculate 61.

addDays (startDate, numberOfDays). Again, this kind of routine might be relevant in systems where schedules are being set as well as forecasting systems, simulators etc. A testbed might include addDays (1999365, 2), which should return 20002.

subtractDays (startDate, numberOfDays). SubtractDays would be relevant in systems similar to those where addDays might be a part of the system functionality.

The same considerations apply to arithmetic routines as well as conversion and extraction routines when date and time representations are compounded. Of particular importance here is the consideration of correctly interpreting the
time 12:00 as midnight or noon when a 12-hour time representation is used.

Date Comparison Functionality

The kinds of routines to be tested here include standard sorting and searching functionality. This kind of processing represents the majority of date usage in software

sort (list, ascending). Given a list of dates, or time-and-date timestamps, returns a list sorted correctly in ascending or descending order, depending on the second parameter.

LessThan (YYYYMMDD, YYYYMMDD). No sorting routine can exist without complementary comparison routines to support it. Comparison routines are at the heart of the entire year 2000 compliance issue. These routines should be tested thoroughly.

5.2 Test Environment.

This involves preparation of a test environment to test the functional characteristics. This can be further categorized as follows:

Device level testing – Testing of a device in an environment isolated from its normal production environment.
System level testing – Testing of a complete system

It would always be preferable to test a device or complete system *in situ*, i.e. in the normal production environment. This may however not always be possible for various reasons – the system cannot be taken off-line, the time to prepare a test setup in the production environment may be excessive, error recovery may not be possible, etc. Subject matter experts should be consulted for preparation of a valid test environment. Some additional guidelines on preparation of a test environment are as follows:

1. If the test environment is a modified production environment, error recovery procedures must be clearly laid out.
2. All data and software where applicable, must be backed up prior to testing.
3. If a separate test environment is being set up, it must be ensured that all hardware models, revision levels of software etc., are exactly the same as the production environment.
4. All external data interfaces must be isolated so as to avoid any clash or discrepancy with any dates from other systems.

5.3 Control Group testing.

Following the setup of a test environment, testing must be carried out using current dates. This will establish the validity of the test environment.

A different kind of control group testing will need to be carried out for post remediation testing. In this case the modified system should first be tested using current dates to establish that no new errors arise.

5.4 Century testing
Following the successful completion of the control group testing , the system should be tested for century compliance based on the test parameters defined earlier.

6 TESTING PROCEDURES

The guidelines will be used for century testing of devices are defined below. These guidelines are based on the four century compliance criteria defined in Section 4. It must be noted that for each individual system all tests may not apply, and that a checklist should be drawn up based on functionality and the specific application that the system is performing.

6.1 *Definitions*

Century date – Jan 01 2000
Leap Year – Year 1996, 2000, 2016
High-risk dates – 12/31/98, 9/9/99, 12/31/99, 2/28/00,2/29/00,3/01/00

6.2 *Testing Guidelines*

6.2.1 Date setting and Representation.

- System can be set to any date in a range e.g. between 1995 and 2005.
- System can be set to dates both in Julian and Gregorian formats where applicable
- System can be set to high risk dates
- System can be re- initialized from cold start using high risk dates

6.2.2 Date Rollover

- System rolls over correctly on high risk dates
- System rolls over correctly both in powered up and powered down states
- System rolls over correctly both in Gregorian and Julian formats where applicable

6.2.3 Date Arithmetic

- System correctly calculates elapsed dates on either side of century rollover
- System correctly calculates days of the week, based on dates
- System correctly computes leap year dates
- System correctly converts between Julian and Gregorian representations

6.2.4 Date Comparison

- System is able to make correct date comparison e.g. 99 < 00
- System is able to correctly sort date fields on both sides of century.

6.2.5 Date Interface

- System is correctly able to pass date values to external devices and systems
- System is correctly able to maintain date information in the upstream/downstream chain